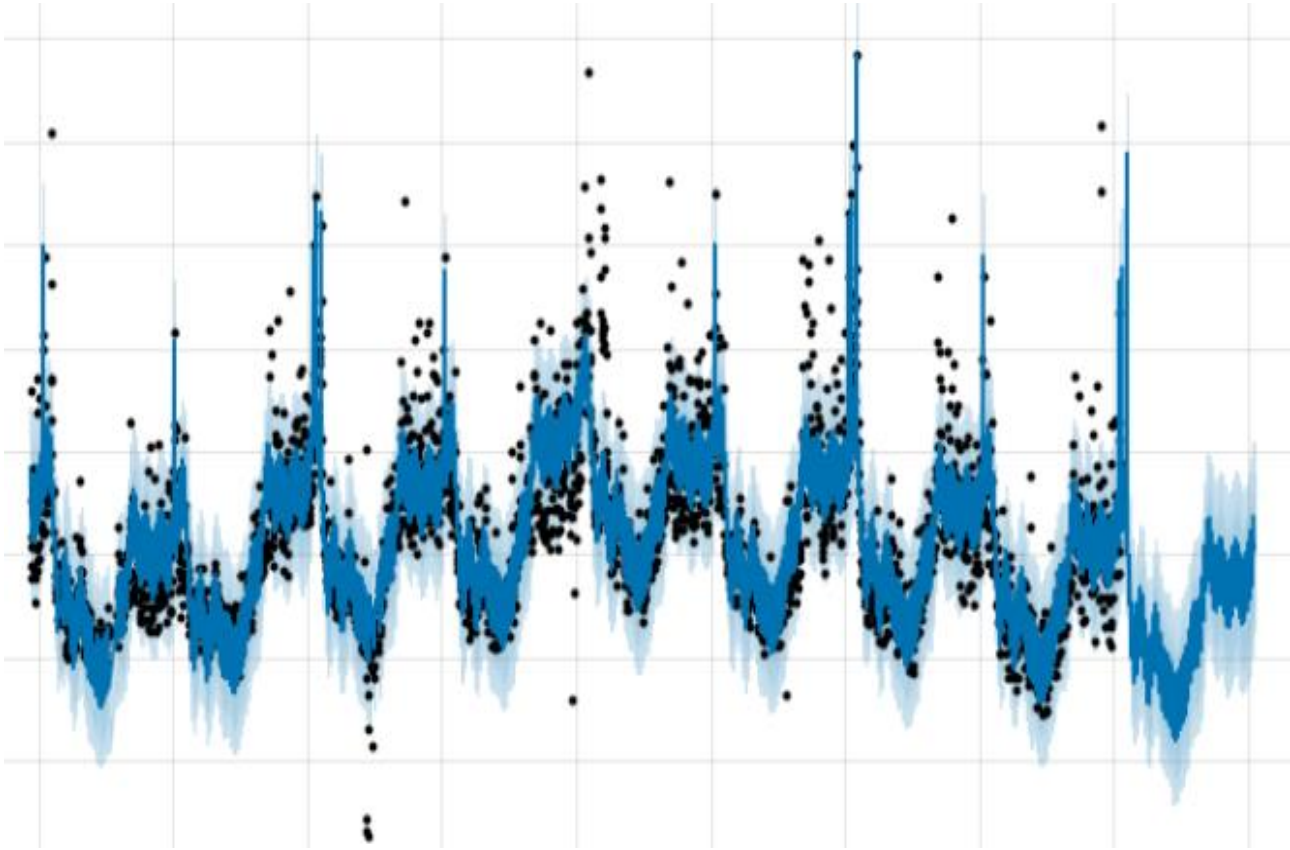
7장 시계열 데이터를 다뤄보자

1. Numpy의 polyfit으로 회귀분석하기
2. Prophet모듈을 이용한 forecast 예측
3. Seasonal 시계열 분석으로 주식 데이터 분석하기
4. Growth Model
5. Prophet 예제
6. 소감



1. Numpy의 polyfit으로 회귀분석하기

In [1]:
```
# 시계열 분석을 하기 위해서는
# 원 데이터의 안정성을 판정하고, 안정한 형태로 변환하고
# 예측 모델을 선정/검증하는 과정에서 깊은 지식을 요구하는데
# 페이스북에서 fbprophet이라는 모듈을 사용하면 간단하게
# 데이터를 가벼운 느낌으로 예측할 수 있습니다
# fbprophet 모듈을 사용하기 위해 pystan과 prophet을 설치합니다
# pip install pystan
# pip install prophet
```

In [2]:
```
# 7-1 Numpy의 polyfit으로 회귀 분석하기
```

In [3]:
```
# 사용할 모듈들을 미리 임포트 했습니다
# pandas_datareader.data as web를 사용하기 위해
# pandas_datareader를 설치해야합니다.
# pip install pandas_datareader
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import pandas_datareader.data as web
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

from fbprophet import Prophet
from datetime import datetime
```

```
ERROR:fbprophet:Importing plotly failed. Interactive plots will not work.
```

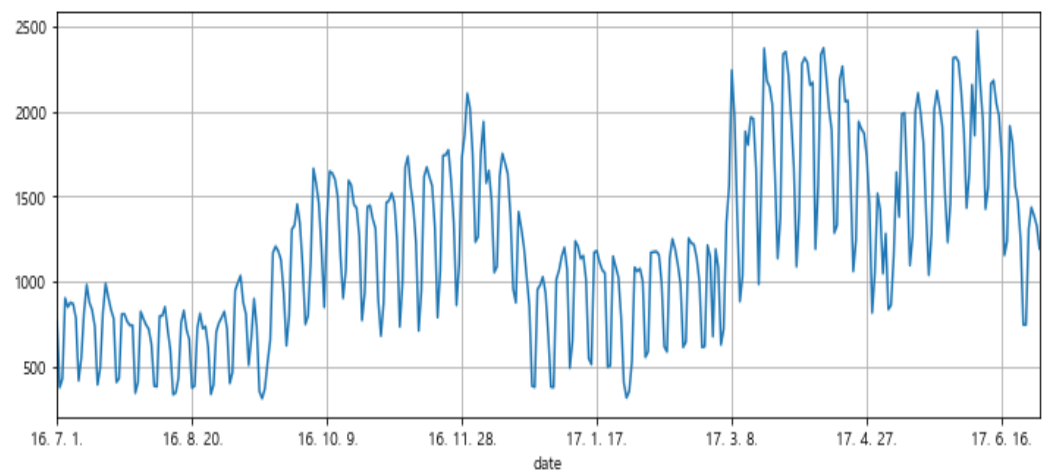
In [4]:
```
# 한글 폰트 설정
path = "c:/Windows/Fonts/malgun.ttf"
import platform
from matplotlib import font_manager, rc
if platform.system() == 'Darwin':
    rc('font', family='AppleGothic')
elif platform.system() == 'Windows':
    font_name = font_manager.FontProperties(fname=path).get_name()
    rc('font', family=font_name)
else:
    print('Unknown system... sorry~~~~')

plt.rcParams['axes.unicode_minus'] = False
```

In [5]:
```
# 필자의 블로그 웹 트래픽 데이터를 읽어왔습니다.
pinkwink_web = pd.read_csv('../data/08. PinkWink Web Traffic.csv',
                           encoding='utf-8', thousands=',',
                           names = ['date','hit'], index_col=0)
pinkwink_web = pinkwink_web[pinkwink_web['hit'].notnull()]
pinkwink_web.head()
```

Out[5]:

date	hit
16.7.1.	766.0
16.7.2.	377.0
16.7.3.	427.0
16.7.4.	902.0
16.7.5.	850.0

In [6]:
```
# 블로그에 사람들이 유입된(블로그 방문수) 정도를 그렸습니다.
pinkwink_web['hit'].plot(figsize=(12,4), grid=True);
```



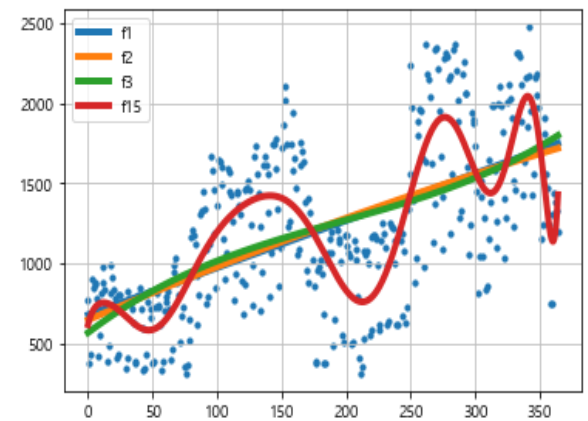
In [7]:
```
# 위의 그래프를 설명할 함수를 찾는 과정 #
#       그래프가 직선일 수도 있고,      #
#      다항식의 곡선일 수도 있는데      #
#    아무튼 이 데이터를 간단한 모델로   #
#    표현하는 작업을 회귀라고 합니다.   #
# 시간축을 만들고
# 웹 트래픽의 자료를 변수에 저장했습니다.
# arange() 특정한 규칙에 따라
#          증가하는 수열을 만듭니다.
# len(pinkwink_web) = 365
# time = np.arange(0, 365) >> 0~364 array생성
# linspace() 선형 간격의 벡터를 생성
time = np.arange(0,len(pinkwink_web))
traffic = pinkwink_web['hit'].values

fx = np.linspace(0, time[-1], 1000)
```

In [8]:
```
# 모델을 1차, 2차, 3차, 15차 다항식으로
# 만들어 표현하고 결과를 확인하기 전에,
# 데이터를 어떤 모델로 표현했을때
# 그 모델의 적합성을 확인하는
# 에러 함수를 정의했습니다.
# sqrt() : 제곱근 결과 리턴(음수x)
def error(f, x, y):
    return np.sqrt(np.mean((f(x)-y)**2))
```

In [9]:
```
# polyfit() 다항식 곡선 피팅
# 1,2,3,15차식의 다항식 곡선을 피팅합니다
# 곡선들의 x,y,z, ... 절편을 구합니다.
# 1차식은 2개, 2차식은 3개, 3차식은 4개, ...
fp1 = np.polyfit(time, traffic, 1)
f1 = np.poly1d(fp1)

f2p = np.polyfit(time, traffic, 2)
f2 = np.poly1d(f2p)

f3p = np.polyfit(time, traffic, 3)
f3 = np.poly1d(f3p)

f15p = np.polyfit(time, traffic, 15)
f15 = np.poly1d(f15p)

print("f01 :", error(f1, time, traffic))
print("f02 :", error(f2, time, traffic))
print("f03 :", error(f3, time, traffic))
print("f15 :", error(f15, time, traffic))
```

```
f01 : 430.85973081109626
f02 : 430.62841018946943
f03 : 429.5328046676293
f15 : 330.4777304274343
```

In [10]:
```
# 1,2,3,15 차 함수를 그린 그래프로 표현했습니다.
plt.figure(figsize=(6,4))
plt.scatter(time, traffic, s=10)

plt.plot(fx, f1(fx), lw=4, label='f1')
plt.plot(fx, f2(fx), lw=4, label='f2')
plt.plot(fx, f3(fx), lw=4, label='f3')
plt.plot(fx, f15(fx), lw=4, label='f15')

plt.grid(True, linestyle='-', color='0.75')

plt.legend(loc=2)
plt.show()
```



2. Prophet모듈을 이용한 forecast 예측

In [11]:
```
# 7-2 Prophet모듈을 이용한 forecast 예측
```

In [12]:
```
# 날짜와 방문수만 따로 데이터 프레임으로 저장했습니다.
# reset_index(inplace=True) 첫번째 열 제거
# to_datetime()를 사용해서 날짜라고 선언했습니다.
# Prophet()를 사용할 때 주기성이 연단위로 있다고 알려줍니다. < ??
df = pd.DataFrame({'ds':pinkwink_web.index, 'y':pinkwink_web['hit']})
df.reset_index(inplace=True)
df['ds'] =  pd.to_datetime(df['ds'], format="%y. %m. %d.")
del df['date']

m = Prophet(yearly_seasonality=True, daily_seasonality=True)
m.fit(df);
```

In [13]:
```
# 60일간의 데이터를 예측합니다.
future = m.make_future_dataframe(periods=60)
future.tail()
```

Out [13]:

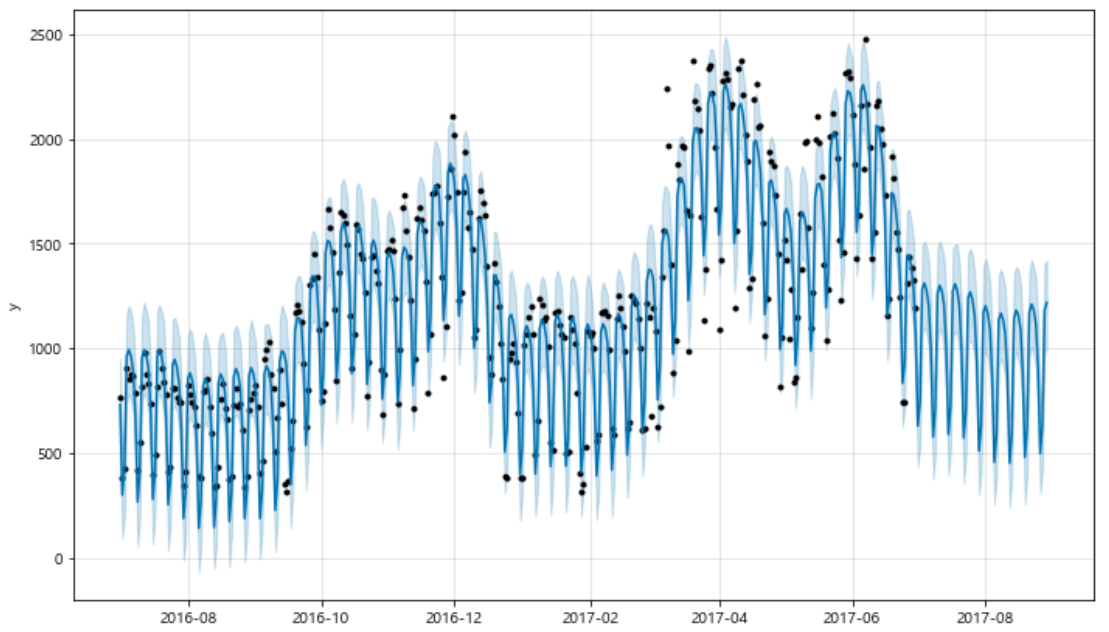
	ds
420	2017-08-25
421	2017-08-26
422	2017-08-27
423	2017-08-28
424	2017-08-29

In [14]:
```
# 예측한 데이터를 변수에 저장하고
# 예측 결과를 데이터 프레임으로 저장합니다.
forecast = m.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

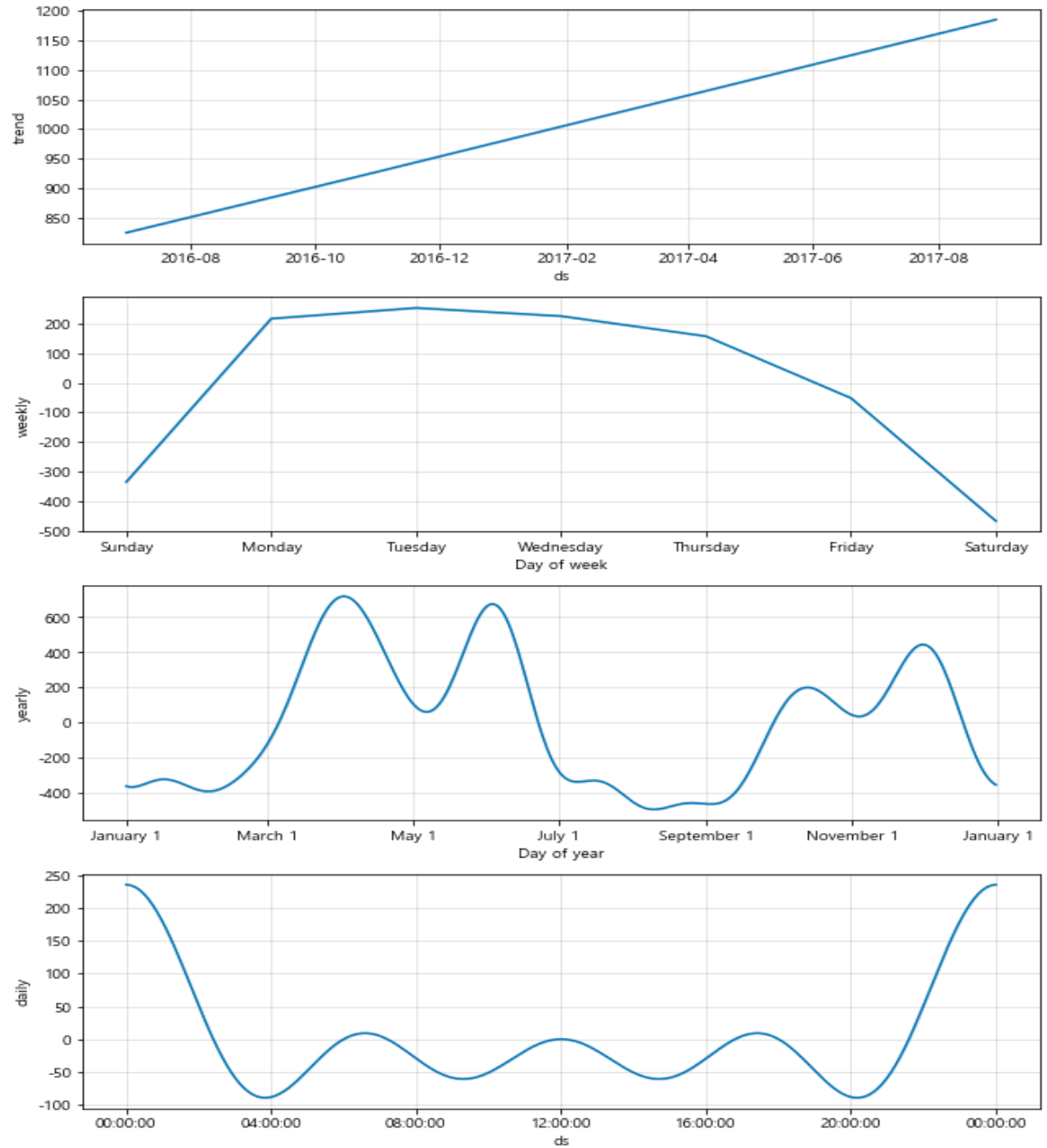
Out [14]:

	ds	yhat	yhat_lower	yhat_upper
420	2017-08-25	911.790860	713.954981	1120.968315
421	2017-08-26	496.917939	304.338282	712.495326
422	2017-08-27	629.458439	421.297214	838.436910
423	2017-08-28	1181.938386	969.367603	1405.117589
424	2017-08-29	1217.864960	1004.163420	1417.931040

In [15]:
```
# 예측한 데이터를 그래프로 그렸습니다.
# 2017년 6월 말 이후 60일간의 예측 결과를
# 그래프로 그렸습니다.
m.plot(forecast);
```



In [16]:
```
# 전체적인 경향을 직선으로 표현할 수 있습니다.
m.plot_components(forecast);
```



3. Seasonal 시계열 분석으로 주식 데이터 분석하기

In [17]:
```
# 7-3 Seasonal 시계열 분석으로 주식 데이터 분석하기
```

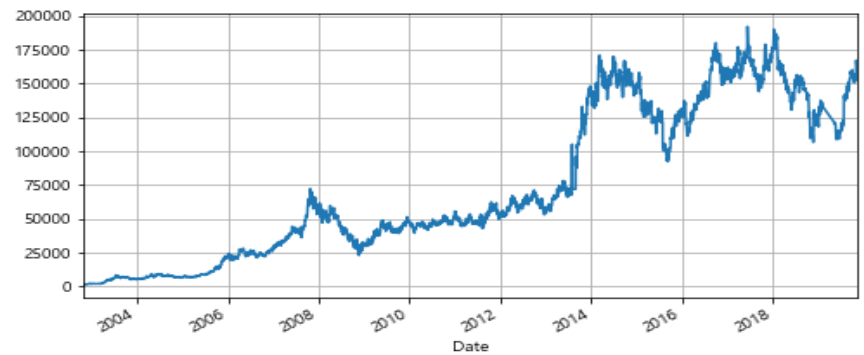
In [18]:
```
# DataReader() 함수를 사용해서 한국 kospi 주가 정보를 받아왔습니다.
# pip install yfinance
# https://docs.google.com/spreadsheets/d/
# 10xaMfpriT6G1d4hw94L6q_ozTOfogilVYb-S2dtw8-8/edit#gid=2134974318
# 받아온 주가 정보는 naver의 주가 정보입니다.
from pandas_datareader import data
import yfinance as yf
yf.pdr_override()

start_date = '2000-1-1'
end_date = '2019-11-05'
naver = data.get_data_yahoo('035420.KS', start_date, end_date)
naver.tail()
```

```
[*********************100%***********************]  1 of 1 completed
```

Out [18]:

Date	Open	High	Low	Close	Adj Close	Volume
2019-10-29	157500.0	158000.0	152000.0	153000.0	153000.0	321007
2019-10-30	152000.0	154500.0	151500.0	154500.0	154500.0	290658
2019-10-31	159000.0	167000.0	157500.0	164000.0	164000.0	1409619
2019-11-01	166000.0	168500.0	164000.0	167000.0	167000.0	720358
2019-11-04	165500.0	167000.0	162000.0	164500.0	164500.0	455199

In [19]:
```
# 네이버의 2001년 1월1일부터 2017년6월30일까지의
# 주가 정보를 받아 간단하게 그래프를 그렸습니다
naver['Close'].plot(figsize=(8,4), grid=True);
```



In [20]:
```
naver.tail()
```

Out [20]:

Date	Open	High	Low	Close	Adj Close	Volume
2019-10-29	157500.0	158000.0	152000.0	153000.0	153000.0	321007
2019-10-30	152000.0	154500.0	151500.0	154500.0	154500.0	290658
2019-10-31	159000.0	167000.0	157500.0	164000.0	164000.0	1409619
2019-11-01	166000.0	168500.0	164000.0	167000.0	167000.0	720358
2019-11-04	165500.0	167000.0	162000.0	164500.0	164500.0	455199

In [21]:
```
# 2019년11월01일 전의 데이터를 잘라 변수에 저장했습니다.
naver_trunc = naver[:'2019-11-01']
naver_trunc.tail()
```

Out [21]:

Date	Open	High	Low	Close	Adj Close	Volume
2019-10-28	152500.0	158000.0	152500.0	157000.0	157000.0	335238
2019-10-29	157500.0	158000.0	152000.0	153000.0	153000.0	321007
2019-10-30	152000.0	154500.0	151500.0	154500.0	154500.0	290658
2019-10-31	159000.0	167000.0	157500.0	164000.0	164000.0	1409619
2019-11-01	166000.0	168500.0	164000.0	167000.0	167000.0	720358

In [22]:
```
# 네이버 주가의 정보를 별도의 데이터 프레임으로 만들었습니다.
df = pd.DataFrame({'ds':naver_trunc.index, 'y':naver_trunc['Close']})
df.reset_index(inplace=True)
del df['Date']
df.tail()
```

Out [22]:

	ds	y
4129	2019-10-28	157000.0
4130	2019-10-29	153000.0
4131	2019-10-30	154500.0
4132	2019-10-31	164000.0
4133	2019-11-01	167000.0

In [23]:
```
# 1년 후의 2020년11월01까지의 정보를 예측했습니다
m = Prophet(daily_seasonality=True)
m.fit(df);
future = m.make_future_dataframe(periods=365)
future.tail()
```

Out [23]:

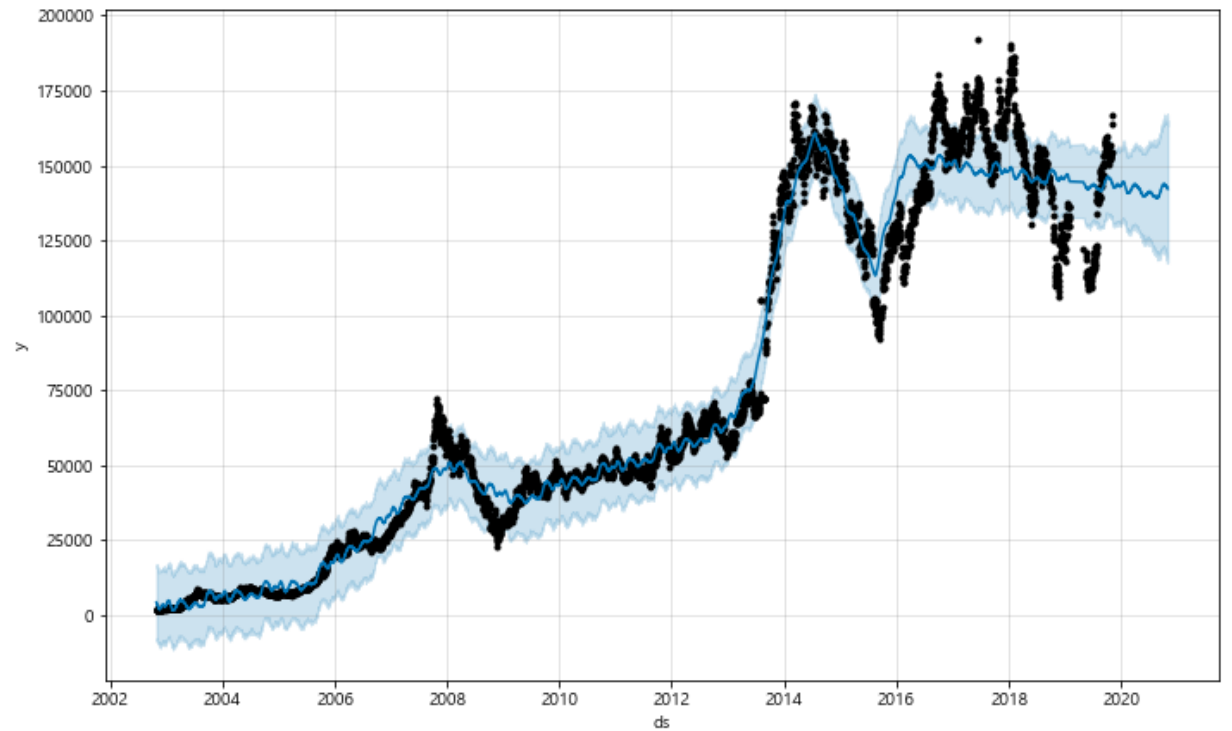
	ds
4494	2020-10-27
4495	2020-10-28
4496	2020-10-29
4497	2020-10-30
4498	2020-10-31

In [24]:
```
# 1년 후의 2020년11월01까지의 정보를 예측했습니다
forecast = m.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

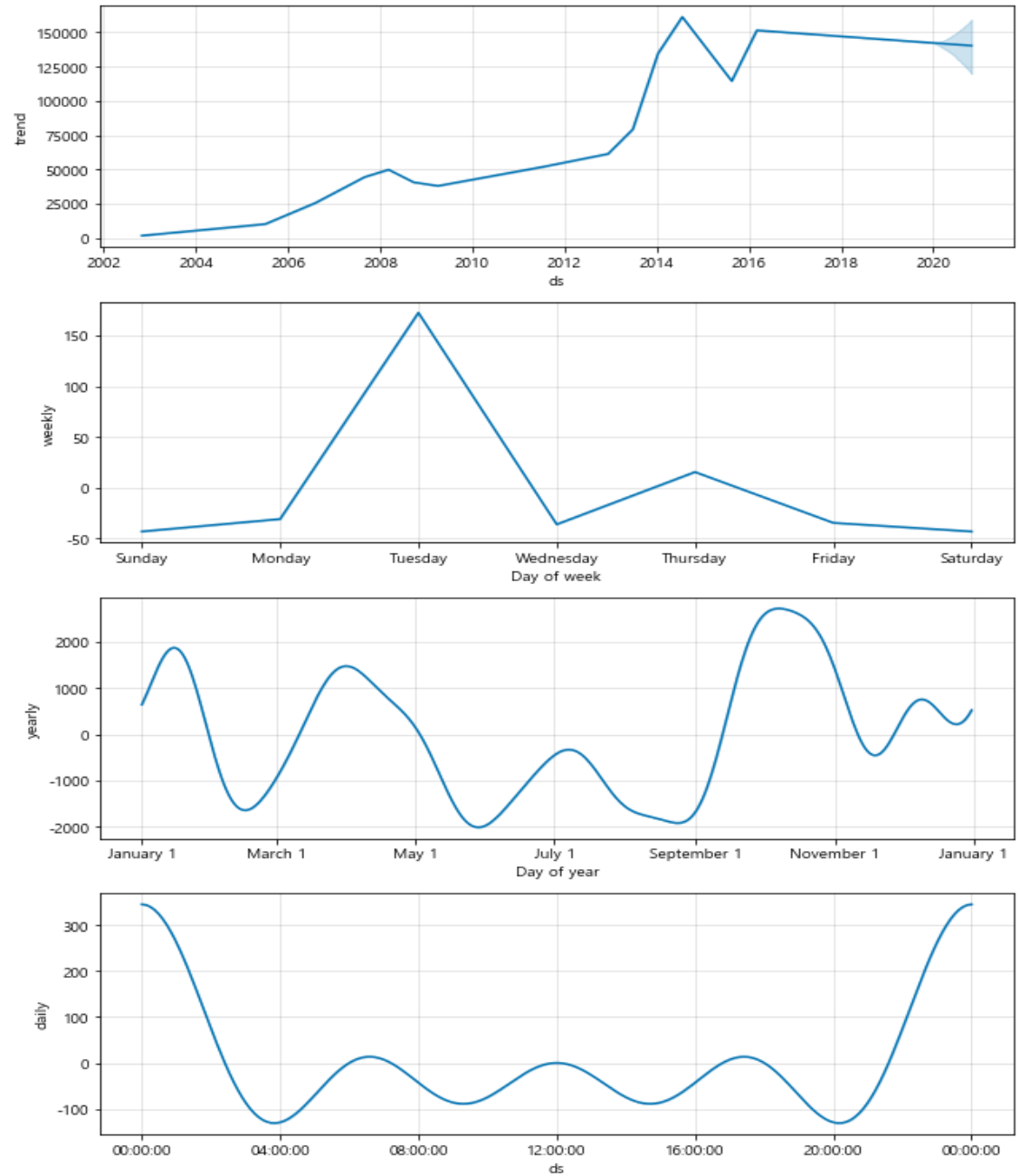
Out [24]:

	ds	yhat	yhat_lower	yhat_upper
4494	2020-10-27	142870.927115	119532.528720	167326.106737
4495	2020-10-28	142551.830201	120067.871354	165082.015827
4496	2020-10-29	142484.625610	117647.341518	166767.899557
4497	2020-10-30	142307.321284	117821.862785	163131.417509
4498	2020-10-31	142164.076542	119528.736247	165132.955049

In [25]:
```
# 예측한 데이터로 그래프를 그렸습니다.
m.plot(forecast);
```

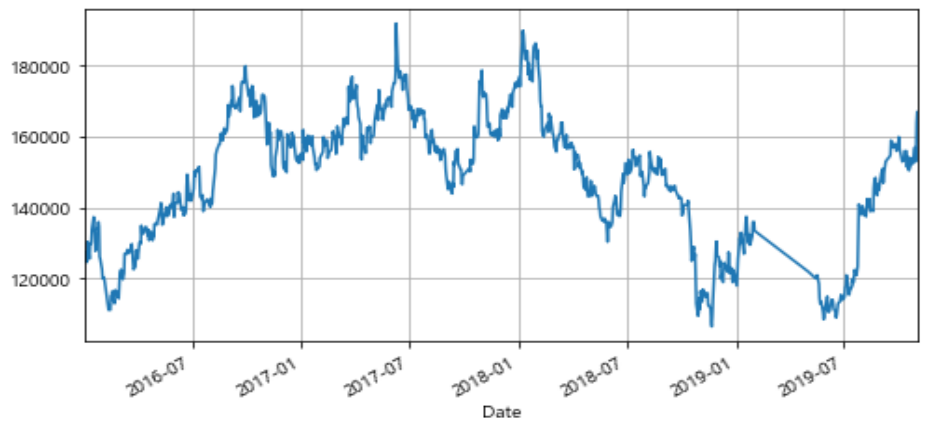


In [26]:
```
# 전체적인 경향을 직선으로 표현했습니다.
m.plot_components(forecast);
```

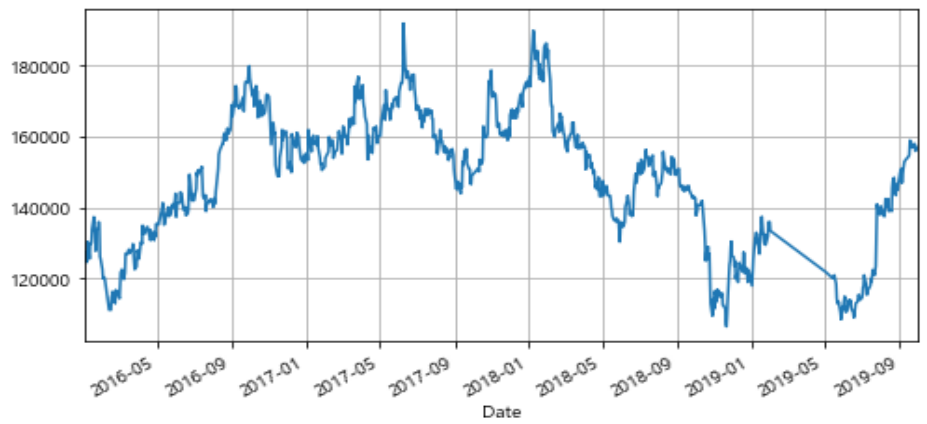


In [27]:
```
# 주가를 가져오는 날짜를 변경해서 다시 저장했습니다.
start_date = '2016-1-1'
end_date = '2019-11-1'
naver = data.get_data_yahoo('035420.KS', start_date, end_date)
naver['Close'].plot(figsize=(8,4), grid=True);
```

```
[**********100%**********]  1 of 1 completed
```



In [28]:
```
# 예측한 데이터와 실제 데이터를 비교하기 위해서
# 사용할 데이터를 2016년1월1일부터 2019년11월1일까지이고,
# 예측용으로 사용할 데이터는 2019년9월31일까지로 저장합니다.
naver_trunc = naver[:'2019-9-30']
naver_trunc['Close'].plot(figsize=(8,4), grid=True);
```



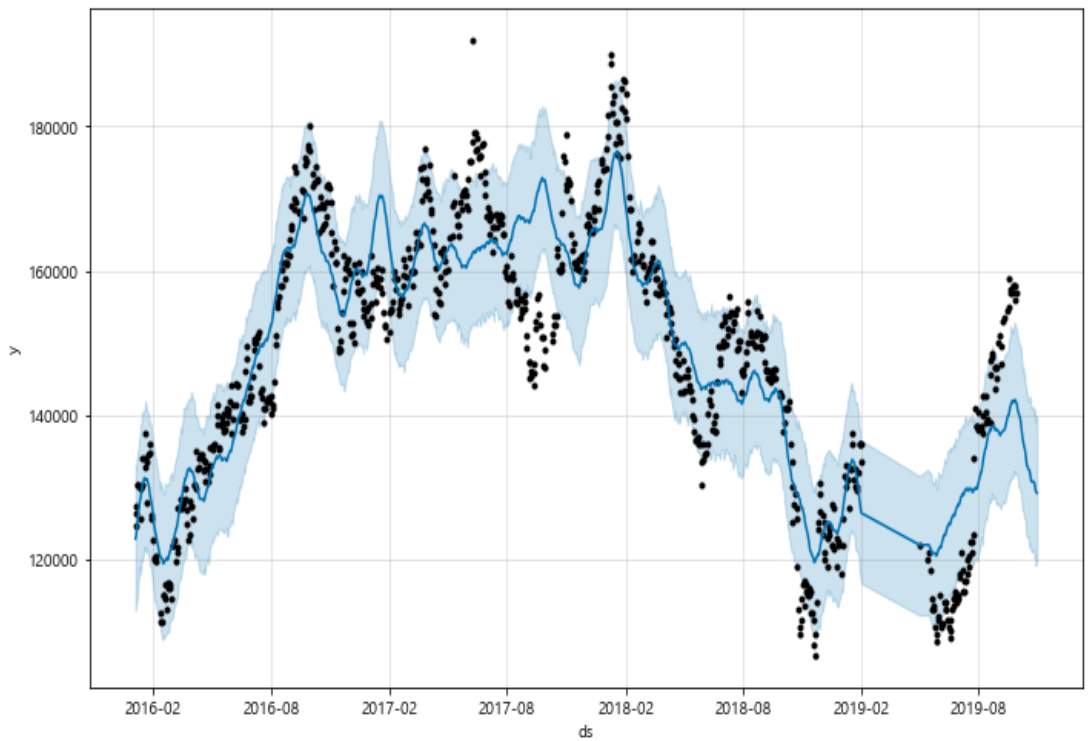
In [29]:
```
# 데이터를 데이터 프레임으로 수정합니다.
df = pd.DataFrame({'ds':naver_trunc.index, 'y':naver_trunc['Close']})
df.reset_index(inplace=True)
del df['Date']
```

In [30]:
```
# 31일간의 데이터를 예측합니다.
m = Prophet(daily_seasonality=True)
m.fit(df);
future = m.make_future_dataframe(periods=31)
future.tail()
```

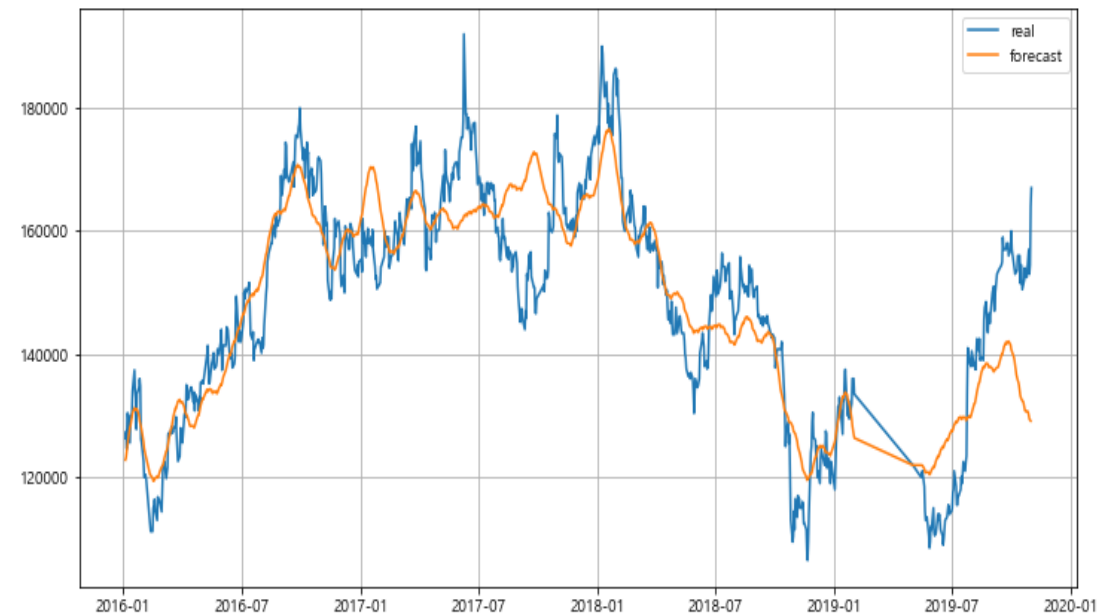
Out [30]:

	ds
872	2019-10-27
873	2019-10-28
874	2019-10-29
875	2019-10-30
876	2019-10-31

In [31]:
```
# 예측한 데이터를 그래프로 그려 확인했습니다
forecast = m.predict(future)
m.plot(forecast);
```



In [32]:
```
# 실제 데이터와 예측한 데이터를 비교했습니다.
plt.figure(figsize=(12,6))
plt.plot(naver.index, naver['Close'], label='real')
plt.plot(forecast['ds'], forecast['yhat'], label='forecast')
plt.grid()
plt.legend()
plt.show()
```



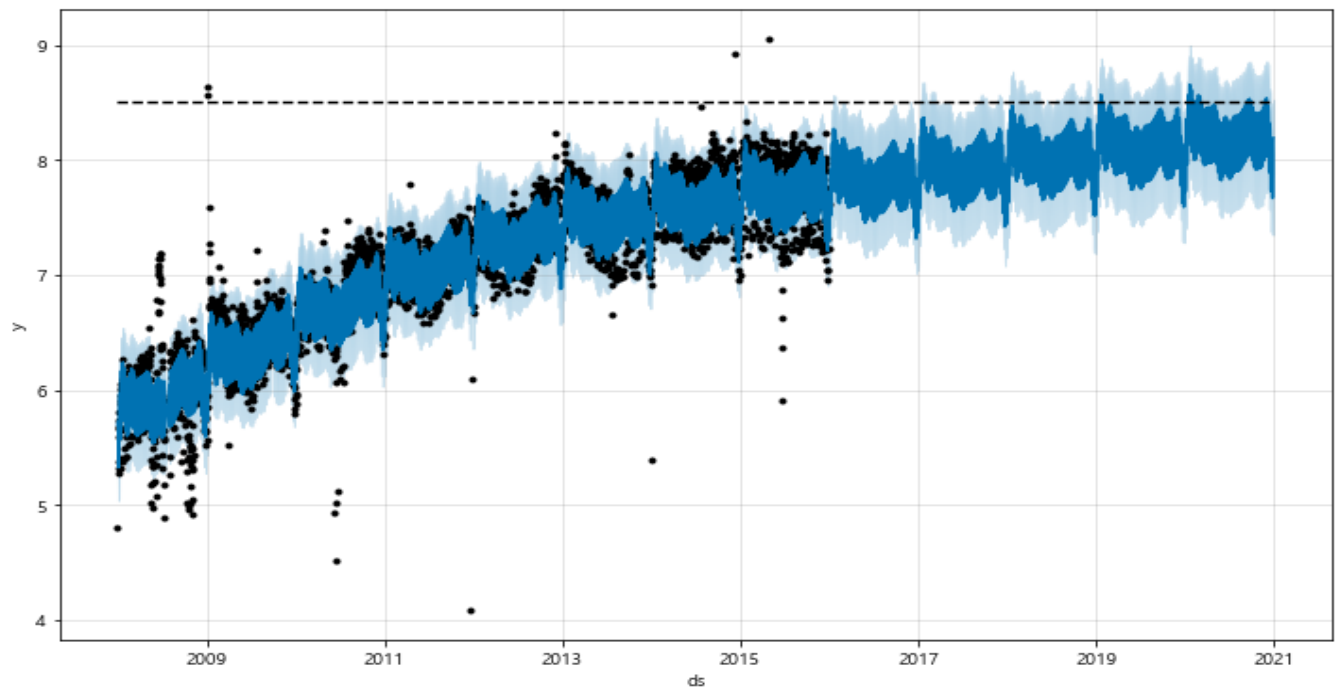
4. Growth Model

In [33]:
```
# 7-4 Growth Model
# prophet 예제
```

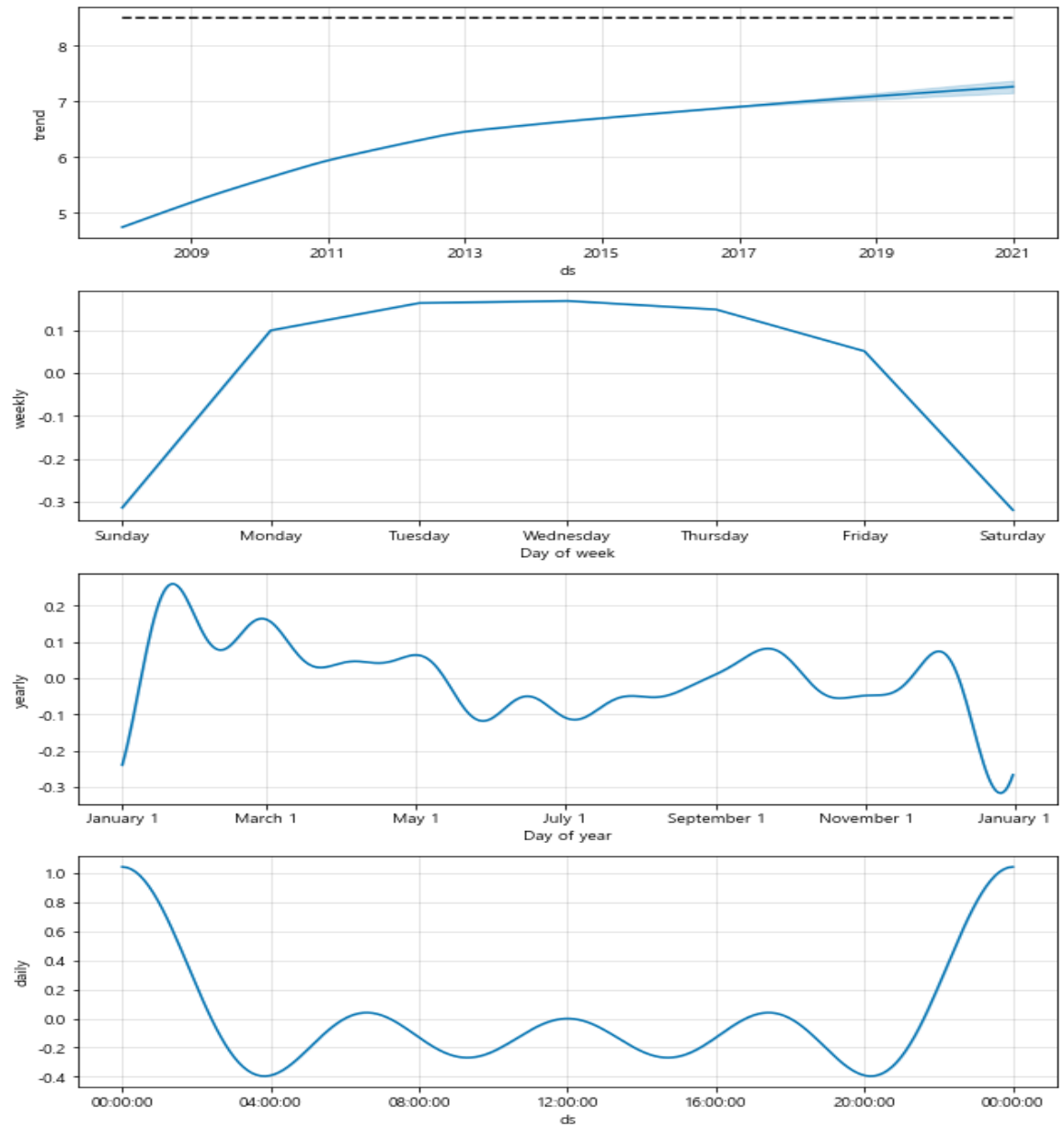
In [34]:
```
# 주기성을 띠면서 점점 성장하는 모습의 데이터를
# 변수에 저장하고 예측함수로 그래프를 그렸습니다.
df = pd.read_csv('../data/08. example_wp_R.csv')
df['y'] = np.log(df['y'])
df['cap'] = 8.5
m = Prophet(growth='logistic', daily_seasonality=True)
m.fit(df)
```

Out[34]:
```
<fbprophet.forecaster.Prophet at 0x1be9d7c6748>
```

In [35]:
```
# 로그함수의 모양을 띕니다.
future = m.make_future_dataframe(periods=1826)
future['cap'] = 8.5
fcst = m.predict(future)
m.plot(fcst);
```



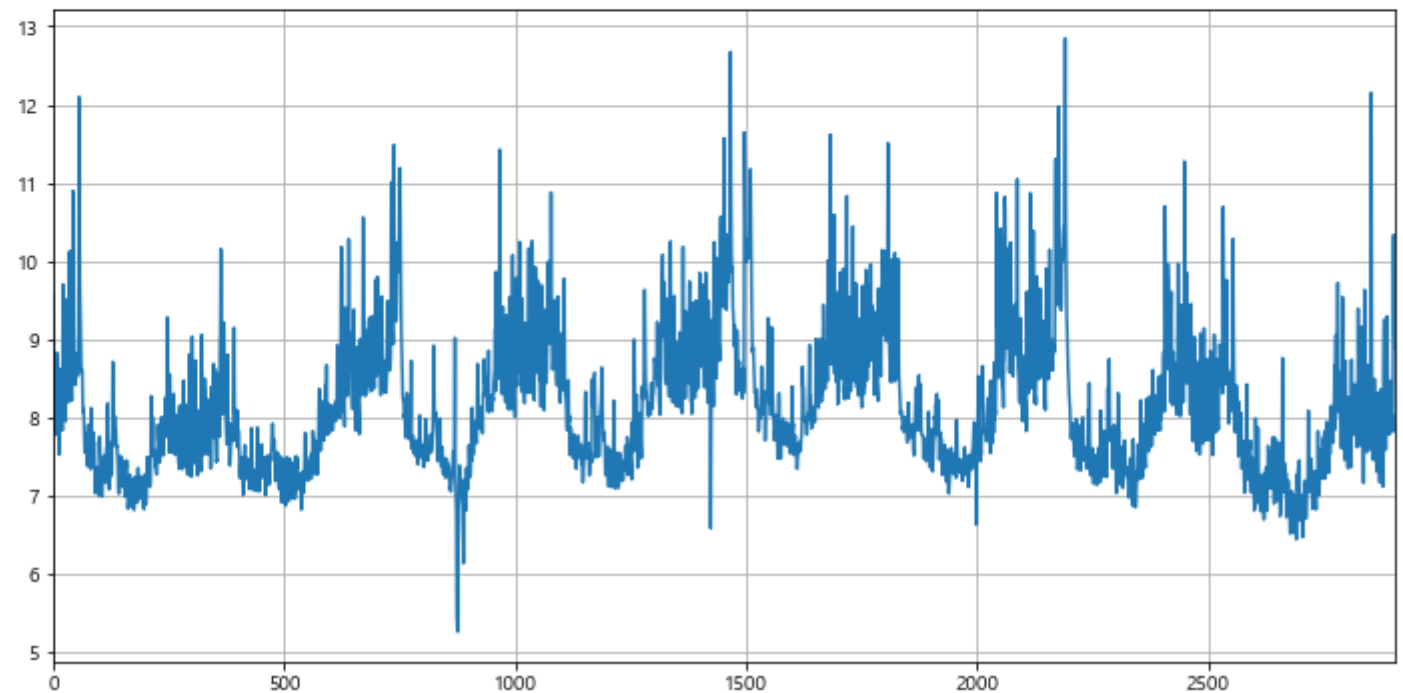
In [36]:
```
# 이런 모양의 데이터도 예측하여 경향을 파악할 수 있습니다.
forecast = m.predict(future)
m.plot_components(forecast);
```



5. Prophet 예제

In [37]:
```
# Prophet예제
df = pd.read_csv('../data/08. example_wp_peyton_manning.csv')
df['y'] = np.log(df['y'])
m = Prophet(daily_seasonality=True)
m.fit(df)
future = m.make_future_dataframe(periods=366)
```

In [38]:
```
# 예측 데이터를 그렸습니다
df.y.plot(figsize=(12,6), grid=True);
```



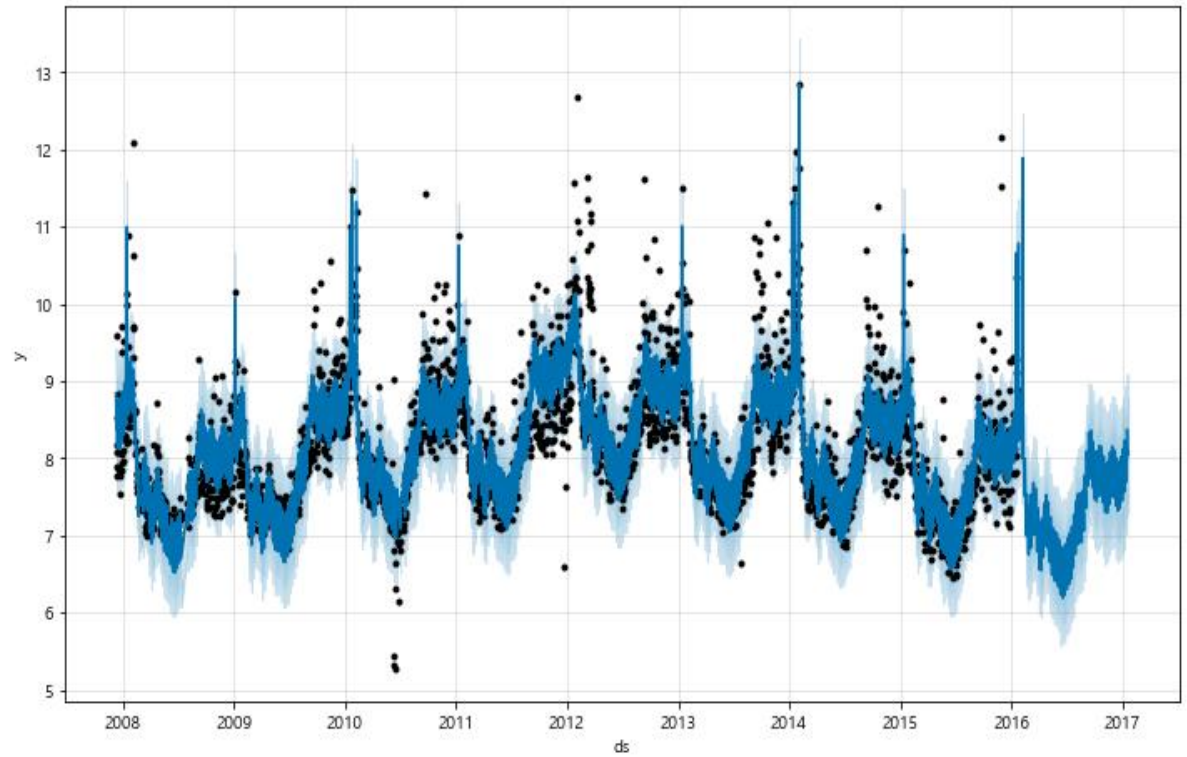
In [39]:
```
# Prophet예제
playoffs = pd.DataFrame({
  'holiday': 'playoff',
  'ds': pd.to_datetime(['2008-01-13', '2009-01-03', '2010-01-16',
                        '2010-01-24', '2010-02-07', '2011-01-08',
                        '2013-01-12', '2014-01-12', '2014-01-19',
                        '2014-02-02', '2015-01-11', '2016-01-17',
                        '2016-01-24', '2016-02-07']),
  'lower_window': 0,
  'upper_window': 1,
})
superbowls = pd.DataFrame({
  'holiday': 'superbowl',
  'ds': pd.to_datetime(['2010-02-07', '2014-02-02', '2016-02-07']),
  'lower_window': 0,
  'upper_window': 1,
})
holidays = pd.concat((playoffs, superbowls))
```

In [40]:
```
# Prophet예제
m = Prophet(holidays=holidays, daily_seasonality=True)
forecast = m.fit(df).predict(future)

forecast[(forecast['playoff'] + forecast['superbowl']).abs() > 0][
        ['ds', 'playoff', 'superbowl']][-10:]
```

Out [40]:

	ds	playoff	superbowl
2190	2014-02-02	1.224116	1.203719
2191	2014-02-03	1.902824	1.458890
2532	2015-01-11	1.224116	0.000000
2533	2015-01-12	1.902824	0.000000
2901	2016-01-17	1.224116	0.000000
2902	2016-01-18	1.902824	0.000000
2908	2016-01-24	1.224116	0.000000
2909	2016-01-25	1.902824	0.000000
2922	2016-02-07	1.224116	1.203719
2923	2016-02-08	1.902824	1.458890

In [41]:
```
# Prophet예제
m.plot(forecast);
```



In [42]:
```
# Prophet예측
m.plot_components(forecast);
```

6. 소감

이번 과제에서는 시계열 데이터에 대한 실습을 진행했습니다.

과제를 진행하면서 시간에 따른 데이터의 변화를 그래프로 그려서 한눈에 볼 수 있었고,
주어진 데이터를 가지고 미래의 데이터를 예측할 수 있다는 점이 매력적이었습니다.

고질적인 문제인 모듈의 설치문제는 제외하고 어려웠던점은 함수의 사용법이었는데,
이를 해결하기 위해 구글링하면서 최대한 설명을 주석으로 옮겨적으려고 노력했습니다.